MANUALE TURBO PASCAL 7.0

INTRODUZIONE - LINGUAGGIO DI PROGRAMMAZIONE

Noi usiamo un linguaggio per comunicare: l'italiano, l'inglese, il francese, il tedesco ecc... Lo sfruttiamo per esprimere i nostri pensieri ad altri esseri umani che lo comprendono, cioè comunichiamo con loro. Il linguaggio di programmazione non è altro che la 'lingua' con cui il programmatore comunica con il computer e gli da istruzioni. Come ogni altra lingua, esso dispone di regole per la sintassi, di parole e di potenzialità che altri non possiedono. Vedremo di capire, con uno sguardo d'insieme la sintassi e le regole del linguaggio Pascal.

Termini specifici

Prima di iniziare la guida vera e propria è doveroso dare una piccola descrizione di alcuni termini usati nel suo corso che non tutti magari conoscono:

File: area di memoria contenente dati;

Estensione: è il tipo di file. Infatti un file di testo, non è la stessa cosa di un file musicale. L'estensione è la sigla che li contraddistingue, che di solito è formata da 3 lettere dopo un punto alla fine del nome. Per visualizzare le estensioni dei file, puoi scaricare il software freeware xp-AntiSpy. Spesso, però, quando il computer non riconosce un tipo di file, appare la sua sigla sotto il nome. Per avere sottomano una tabella con le estensioni principali, puoi scaricare da questo sito 'Guida alle estensioni', nella sezione Appunti, sottosezione Pascal;

Codice sorgente o sorgente: è il file in cui viene scritto il programma con un linguaggio di programmazione;

Compilatore: è il software che converte le istruzioni contenute nel sorgente, sottoforma di testo, in un programma vero e proprio, un'applicazione in formato eseguibile (*.exe o altre volte anche *.com). Per compilare i tuoi programmi scritti in Pascal avrai bisogno di un compilatore adatto: puoi scarica da questo sito i compilatori Turbo Pascal 7.0 o Bloodshed DevPascal nella sezione Downloads, sottosezione Compilatori;

Parola riservata: di solito le parole riservate vengono evidenziate dal compilatore in modo differente rispetto alle altre. Infatti queste parola sono importanti perchè usate per definire delle istruzioni particolari. Pertanto il loro nome è inutilizzabile come variabile.

Libreria: insieme di metodi, istruzioni, operatori, variabili, costanti, classi, procedure o funczioni raccolte in un solo file, che può essere richiamato da un programma Pascal per usufruire delle operazioni che in esso sono descritte;

Implementazione: implementare una procedura o una funzione vuol dire definirne il corpo, ossia le istruzioni e le operazioni che la compongono e ne definiscono il funzionamento

1 LE FASI DELLA PROGRAMMAZIONE

Le fasi di lavoro che devono essere affrontate durante la programmazione sono, come indica la figura 1.1, le seguenti:

- *editing* del programma
- *compilazione e link*
- *esecuzione*

La fase di *editing* è costituita dalla scrittura del programma mediante un sistema di videoscrittura detto *editor*. Il programma ottenuto dopo questa fase si chiama *programma sorgente*.

La *compilazione*, eseguita da un pacchetto software che si chiama *compilatore*, traduce il programma sorgente, in *programma oggetto*, ovvero programma scritto in linguaggio macchina ed eseguibile direttamente dal sistema operativo dell'elaboratore.

Spesso il programma sorgente fa riferimento a parti del codice che non vengono scritte dal programmatore ma che sono reperibili nelle librerie di sistema. Queste parti di codice vengono collegate dal programma nella fase di *link* da una serie di programmi indicati con il termine collettivo *linker.*

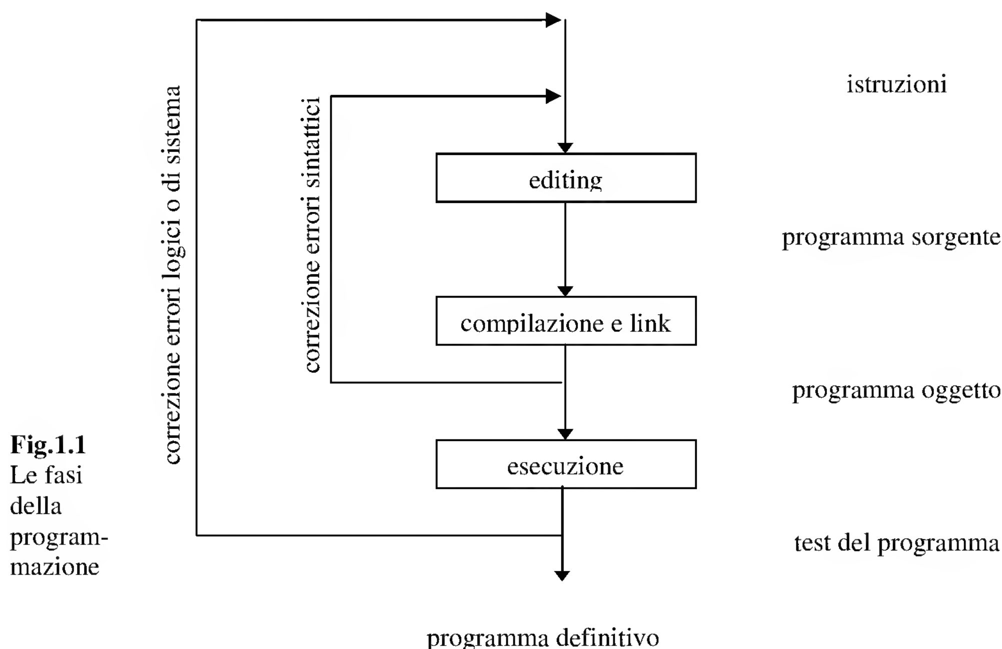


Fig.1.1 Le fasi della programmazione

Nel Turbo Pascal *compilazione e link* sono eseguiti per mezzo di un unico comando. Durante queste fasi, oltre a tradurre il programma nel corrispondente programma oggetto, vengono anche trovati e segnalati *gli errori di carattere sintattico* presenti nel programma. Se nella fase di compilazione vengono evidenziati degli errori, il codice oggetto non può essere prodotto e dobbiamo quindi tornare alla fase di editing per correggerli.

Durante la fase di esecuzione il programma oggetto viene caricato in memoria centrale ed eseguito. In questa fase vengono individuati altri tipi di errori che non sono di tipo sintattico: mancanza di memoria sufficiente per l'esecuzione del programma, o la divisione di un numero per zero. In questi casi l'esecuzione viene interrotta. Sempre durante l'esecuzione il programmatore stesso ha la possibilità di verificare se il programma opera correttamente oppure sono stati commessi errori logici, per cui i risultati ottenuti non sono quelli desiderati.

Un software che gestisce le tre fasi della programmazione, collegando direttamente i vari ambienti, viene chiamato *ambiente di sviluppo integrato. Il Turbo Pascal è un ambiente di sviluppo integrato.*

Talvolta può essere utile salvare il programma oggetto detto anche eseguibile oltre che il programma sorgente. Il file oggetto permette di eseguire direttamente il programma, non

necessariamente sullo stesso computer usato per scrivere il sorgente e senza utilizzare l'ambiente del Turbo Pascal. Per salvare l'eseguibile, al momento delle compilazione scegliere *Make* invece di Compile (oppure premere F9). Prima di salvare accertarsi che nel menu Compile compaia l'opzione Destination Disk se si vuole salvare sul dischetto.

2 STRUTTURA DEI PROGRAMMI

Iniziamo lo studio del Pascal osservando il listato 2.1, in cui possono essere evidenziate alcune caratteristiche comuni alla struttura di ogni programma.

La parola chiave[1] **program**, che apre *l'intestazione*, è seguita dal nome che abbiamo assegnato al programma, nel caso in esame si chiama *paese*.

Fig. 2.1 Il primo programma in Pascal

```
Program paese
Begin
    Write('Tre');
    Write(' casettine');
    Write(' dai tetti aguzzi');
end.
```

Il *corpo* del programma è compreso tra la parola chiave *begin* e la parola chiave *end* seguita da un punto. Il corpo del programma è composto da una serie di istruzioni *write* che vengono eseguite sequenzialmente. Ogni istruzione termina con il carattere punto e virgola. Nll'esecuzione delle istruzioni, il Pascal non distingue tra maiuscole e minuscole. Ogni istruzione write permette la stampa su video di ciò che è racchiuso tra parentesi tonde e apici. Il risultato dell'esecuzione del programma di fig.2.1 è la stampa su video delle seguente frase:

Tre casettine dai tetti aguzzi

Se desideriamo che ogni parola del listato sia scritta su una riga a parte, dobbiamo usare *writeln* in luogo di *write.* Il risultato sarà

Tre
casettine
dai tetti aguzzi

Se nell'istruzione write compare una frase con l'apostrofo, l'apostrofo va sostituito con un apostrofo doppio. Vedi fig.2.2.

Fig. 2.2 Il primo programma in Pascal

```
Program paese
Begin
    Write('Tre');
    Write(' casettine');
    Write(' dall''aria tranquilla');
end.
```

[1] Con *parola chiave o riseravta* si intende una parola a cui il compilatore attribuisce un'interpretazione univoca e determinata riservandola a un uso specifico

Concludendo la struttura generale di un programma Pascal è:

```
program nome_programma;
begin
    istruzione1;
    istruzione2;
    ...
    istruzioneN;
end.
```

3 VARIABILI E ASSEGNAMENTI

Supponiamo di voler calcolare l'area di un rettangolo di base 3 e altezza 7. Osserviamo in fig. 3.1 il listato che risolve il problema.

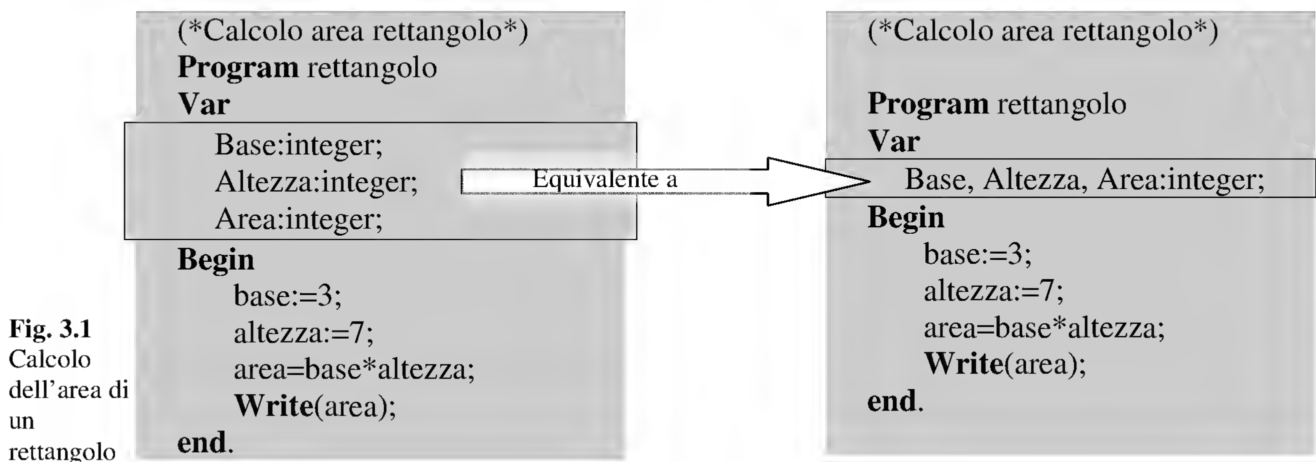


Fig. 3.1 Calcolo dell'area di un rettangolo

Per rendere evidente la funzione espletata dal programma abbiamo inserito un commento. I commenti possono essere inseriti tra i caratteri (*……*) oppure tra graffe.{….}.

Le dichiarazioni di variabili vengono inserite dopo la parola chiave **var** e prima della parola chiave **begin.** Anche le dichiarazioni devono terminare con un punto e virgola. Ogni singola dichiarazione è costituita da un identificatore di variabile, seguito da due punti e dalla parola chiave **integer** che specifica che la variabile è di tipo intero. Il *nome* di una variabile la identifica, il suo *tipo* ne definisce la dimensione e l'insieme delle operazioni che vi si possono effettuare. Il Turbo Pascal riserva per gli *integer* uno spazio di due byte, il che consente di lavorare su interi che vanno da –32768 e +32767. Tra le varie operazioni permesse tra integer vi sono +,-,*.

L'istruzione `base:=3` assegna alla variabile `base` il valore 3.

Il risultato dell'esecuzione del programma in fig.3.1 è il numero 21 scritto sullo schermo.

Se si vuole rendere più chiaro il risultato si possono stampare anche la base e l'altezza, oltre all'area e far precedere ogni valore da una descrizione. Vedi fig. 3.2. Nell'istruzione *write* si inserisci la descrizione tra apici seguita da una virgola e dal nome della variabile.

```
(*Calcolo area rettangolo*)
Program rettangolo
Var
   Base, Altezza, Area:integer;
Begin
     base:=3;
     altezza:=7;
     area=base*altezza;
     write('Base:   ', base);
     write('         Altezza:   ', altezza)
     Write('         Area:   ', area);
end.
```

Fig. 3.1 Calcolo dell'area di un rettangolo

L'esecuzione dà il seguente risultato sullo schermo:

```
Base:  3        Altezza:  7        Area:  21
```

Mentre integer è una parola chiave del Pascal, rettangolo, base, altezza e area sono identificatori scelti da noi a nostra discrezione.

Esistono comunque delle regole da rispettare nella costruzione degli identificatori:

- devono iniziare con una lettera o con un carattere di sottolineatura(_)
- possono contenere solo lettere non accentate, cifre e _.

Esempi di identificatori validi: nome1, cognome_nome, volume, eta.

Esempi di identificatori non validi: 12nome, cognome-nome,vero?, età. Si noti vhe Volume, VOLuMe e voLUme fanno riferimento allo stesso identificatore.

Oltre a rispettare le regole precedentemente enunciate, un identificatore non può

- essere una parola chiave del linguaggio
- essere uguale a uno degli identificatori usati precedentemente.

Concludendo la struttura di un programma Pascal prevede che le variabili siano dichiarate dopo la parola chiave **var** e prima di **begin:**

```
program nome_programma;
var
    dichiarazioni_di_variabili

begin
    istruzione1;
    istruzione2;
    …
    istruzioneN;
end
```

4 COSTANTI

Quando un certo valore non viene modificato nel corso del programma è opportuno, soprattutto se è utilizzato in modo ricorrente, rimpiazzarlo con un nome simbolico. Per farlo dobbiamo definire un *identificatore di costante* e fargli corrispondere il valore desiderato. La dichiarazione di costante va definita prima del **begin** e dopo la parola chiave **const.**

```
(*Calcolo area rettangolo*)
Program rettangolo
const
    BASE=3;
    ALTEZZA=7;

var Area:integer;
Begin
    area=BASE*ALTEZZA;
    write('Base:   ', BASE);
    write('        Altezza:   ', ALTEZZA)
    Write('        Area:   ', area);
end.
```

Fig. 4.1 Calcolo dell'area di un rettangolo

La stessa definizione di costante implica che il suo valore non può essere modificato. Cioè posso utilizzare l'identificatore e patto che su di esso non venga mai effettuato un assegnamento. Per maggior ciarezza scegliamo di indicare un identificatore di costante con caratteri tutti maiuscoli. (BASE, ALTEZZA).

In sintesi, l'uso delle costanti migliora due parametri classici di valutazione dei programmi: flessibilità e facilità di manutenzione. Nel Turbo Pascal le dichiarazioni di costanti e variabili possono apparire in un ordine qualsiasi, fermo restando che devono seguire l'intestazione del programma e precedere il *begin*.

5 IMMISSIONE ED EMISSIONE DI DATI

I programmi descritti negli esempi precedenti non calcolano l'area di un qualsiasi rettangolo ma soltanto di quello che ha per base 3 e per altezza 7. Per rendere il programma di applicazione più generale si deve permettere a chi lo utilizza di immettere i valori della base e dell'altezza: in questo modo l'algoritmo calcolerà l'area di un qualsiasi rettangolo. Per leggere i dati immessi dall'utente si utilizza l'istruzione:

```
read(base);
```

che fa sì che il sistema attenda l'immissione di un dato da parte dell'utente.

Fig. 5.1 Calcolo dell'area di un rettangolo

```
(*Calcolo area rettangolo*)
Program rettangolo
Var
   Base, Altezza, Area:integer;
Begin
    Write('Valore base: ');
    readln(base);
    write('Valore altezza: ');
    readln(altezza);
    area=base*altezza;
    writeln;
    write('Base:   ', base);
    write('        Altezza:   ', altezza)
    Write('        Area:   ', area);
end.
```

La coppia di istruzioni

```
Write('Valore base: ');
readln(base)
```

avrà l'effetto di far comparire sullo schermo la scritta

```
Valore base…:_
```

In questo istante l'istruzione read attende l'immissione di un valore seguito da <INVIO>.
Se l'utente digiterà 15 seguito da <INVIO>

```
Valore base…: 15 <INVIO>
```

Questo dato verrà assegnato alla variabile base.

base
15

L'istruzione read può leggere più valori in ingresso. Per esempio

```
read(base, altezza);
```

i valori immessi saranno separati da uno spazio e e verranno assegnati rispettivamente alle variabili base e altezza.

Analogamente si possono stampare più variabili con una sola istruzione *write.*

Write('Base: ', base,   'Altezza: ', altezza, '   Area: ', area)

Genererà

Base: 3   Altezza: 7   Area: 21

Si può anche definire il numero di caratteri riservati per la visualizzazione di un valore:

Write('Base: ', base:5,   'Altezza: ', altezza:5, '   Area: ', area:5)

6 ESPRESSIONI ARITMETICHE

Un'espressione aritmetica è costituita da un insieme di variabili e costanti numeriche connessi da operatori aritmetici. In figura 6.1 sono riportati gli operatori aritmetici che, applicati agli interi, ritornano valori interi. In particolare nella figura vengono evidenziate le priorità in ordine dalla più alta alla più bassa.

Fig. 6.1 Gerarchia degli operatori aritmetici

-(negazione, operatore unario)
*(moltiplicazione), **div**(divisione ntera), **mod**(modulo)
+(somma, -(sottrazione)

Attenzione: l'assegnamento aun intero di un valore non compreso nell'intervallo previsto, da –32768 a +32767, genera effetti difficilmente prevedibili. Si ha quello che si definisce un *overflow* (traboccamento): l'operatore genera un valore intero che in binario non è rappresentabile con due byte, il bit che costituisce la parte più rappresentativa del numero viene perciò perso e il risultato è completamente errato. Vedremo più avanti che il Turbo Pascal mette a disposizione degli altri tipi di dato che permettono di gestire valori più grandi.

7 FUNZIONI E PROCEDURE PREDEFINITE

Una funzione, analogamente a quanto definito in matematica, è uno strumento che, a partire da uno o più valori presi in input, restituisce un valore associato ai primi in modo univoco.
Più avanti impareremo a creare le nostre funzioni; esistono però delle funzioni predefinite o standard, già pronte all'uso, che il linguaggio mette a disposizione del programmatore. Da questo punto di vista non interessa come il compito affidato alla funzione venga svolto, basta sapere cosa deve esserle passato e cosa restituisce in uscita. Un esempio di funzione predefinita è abs, che applicata a un numero ne restituisce il valore assoluto. Il risultato restituito da una funzione può essere inserito all'interno di un'espressione.
Altre funzioni disponibili in Pascal sono:

pred(num)	restituisce il predecessore di num
succ(num)	restituisce il successore di num
sqr(num)	restituisce il quadratodi num
random(num)	restituisce un numero intero casuale compreso tra 0 e num-1

Nel listato 7.1 si ha un esempio di utilizzo della funzione random per simulare al computer il lancio di due dadi.

```
(*Esempio di utilizzo della funzione random*)
Program lancio_dadi
var a, b, totale:integer;

Begin
     Writeln;
   a:=random(6)+1;
   b:=random(6)+1;
   totale:=a+b;
   writeln ('Lancio:  ', totale)
end.
```

Fig. 7.1 Utilizzo della funzione random(n)

Osserviamo che iterando più volte l'esecuzione del listato otteniamo sempre lo stesso risultato. Questo perché la funzione random genera numeri solo apparentemente casuali, in realtà determinati a partire da un valore iniziale (detto *seme*) memorizzato nella variabile di sistema RandSeed. Per modificare questo valore occorre utilizzare l'istruzione

```
Randomize;
```

che modifica il valore del seme.
Per avere lanci di dadi sempre diversi occorrerà quindi inserire all'inizio del programma:

```
begin
  randomize;
  writeln;
```

Le procedure, come le funzioni, svolgono un compito specifico ma, a differenza di queste, non restituiscono direttamente un valore in uscita e non possono così essere utilizzate all'interno di espressioni. Vengono richiamate, come abbiamo fatto con randomize, specificandone il nome seguito da un punto e virgola; per esempio l'istruzione

```
Clrscr;
```

invoca la procedura clrscr (*CLeRaSCReen*) che pulisce completamente lo schermo e posiziona il cursore nell'angolo in alto a destra dello stesso. Un'altra procedura molto utilizzata è

```
gotoXY(c, r);
```

che sposta il cursore in una certa posizione dello schermo.
L'istruzione

```
gotoXY(25, 10)
```

sposta il cursore nella decima riga in corrispondenza della venticinquesima colonna.
Infine, se il computer con cui lavoriamo dispone di uno schermo a colori sono interessanti le procedure predefinite

```
TextColor(n);
```

che modifica il colore con cui i dati vengono visualizzati sullo schermo; e

```
TextBackground (n);
```

che modifica il colore dello sfondo su cui viene visualizzato il testo. Per esempio, le istruzioni

```
textcolor(14);
textbackground(4);
writeln('Prova testo a colori');
```

visualizzano sulla schermo la scritta Prova testo a colori in caratteri gialli su sfondo rosso. I colori disponibili dipendono dal computer; nella tabella 7.2 sono riportati i codice più comuni.

Colore	Valore	Colore	Valore
Nero	0	Grigio scuro	8
Blu	1	Blu chiaro	9
Verde	2	Verde chiaro	10
Turchese	3	Turchese chiaro	11
Rosso	4	Rosso chiaro	12
Magenta	5	Magenta chiaro	13
Marrone	6	Giallo	14
Grigio chiaro	7	Bianco	15

Fig. 7.2 Codici dei colori

Le procedure ClrScr, TextColor, TextBackground e GotoXY non vengono collegate automaticamente ai nostri programmi come avviene per le funzioni viste in precedenza. Si deve quindi richiedere esplicitamente questa connessione al sistema, inserendo dopo l'intestazione del programma e prima della dichiarazione di costanti e variabili

```
Uses crt;
```

dove crt (*Cathode Ray Tube*) è il nome della libreria (o modulo) contenente l'insieme delle funzioni e delle procedure per la gestione del video. In precedenza abbiamo potuto utilizzare direttamente le funzioni esaminate perché queste, come molte altre funzioni e procedure, sono contenute nel modulo *system* che viene automaticamente collegato al programma dal compilatore.

Una dichiarazione uses può riguardare un modulo oppure una lista di moduli: in quest'ultimo caso gli elementi della lista devono essere separati da una virgola.

La possibilità di includere diversi moduli utilizzando le relative funzioni e procedure ci porta a definire una struttura sintattica del programma più generale:

```
program nome_programma;
uses lista_di_moduli;
parte_dicharativa

begin
    istruzione1;
    istruzione2;
    ...
    istruzioneN;
end
```

Altri moduli del Turbo Pascal sono Printer per l'uso della stampante e Graph per la grafica.

Anche read e write sono in realtà procedure standard. La procedura read accetta in ingresso le variabili in cui immettere i valori letti e write accetta i commenti e le variabili da scrivere.

I valori che le funzioni e le procedure accettano tra parentesi tonde sono detti *parametri*; si noti che ClrScr non accetta alcun parametro in ingresso, mentre GotoXY ne accetta due.

8 STRUTTURE DI CONTROLLO E DECISIONE

8.1 If Then Else

Quando si desidera eseguire un'istruzione solo al verificarsi di una certa condizione, si utilizza l'istruzione if. La sintassi è la seguente:

```
if espressione_logica then
    istruzione1
[else
    istruzione2];
```

Se espressione_logica risulta vera, viene eseguita istruzione1, se risulta falsa viene eseguita istruzione2. Il ramo *else* è opzionale come evidenziato dalle parentesi quadre.

Un'*espressione logica* è caratterizzata dal fatto che la sua valutazione può generare soltanto uno dei due valori logici TRUE o FALSE.

Attenzione: l'istruzione che precede l'else non deve essere chiusa da un punto e virgola, altrimenti il compilatore genera un errore.

Istruzione1 e istruzione2 possono essere istruzioni *semplici* o *composte*. Un'istruzione *semplice* è un'unica istruzione. Un'istruzione *composta*, detta anche *blocco*, è costituita da un insieme di istruzioni inserite tra le parole chiave begin – end che il compilatore tratta come se fossero un'unica istruzione. Il costrutto if è a sua volta un'istruzione e quindi può comparire all'interno di un altro if, nel ruolo di istruzione. Quando ciò si verifica si parla di if annidati:

```
if i<100 then
    if i>0 then
        writeln ('minore di 100 e maggiore di zero');
```

Il secondo controllo i>0 viene effettuato solo se il primo i<100 ha dato esito positivo.

Aggiungiamo al nostro esempio un ramo else:

```
if i<100 then
    if i>0 then
        writeln ('minore di 100 e maggiore di zero')
    else
        writeln ('minore di 100 ma non maggiore di zero');
```

L'else che abbiamo aggiunto si riferisce al secondo if cioè a quello più interno. Per fare in modo che l'else si riferisca al primo if bisogna usare un blocco begin-end:

```
if i<100 then
    begin
    if i>0 then
        writeln ('minore di 100 e maggiore di zero')
    end
else
        writeln ('maggiore o uguale a 100');
```

Un'ulteriore modifica al programma è la seguente:

```
1   if i<100 then
2         if i>0 then
3                 writeln ('minore di 100 e maggiore di zero')
4         else
5                 if i=0 then
6                       writeln('ugule a zero')
7                 else
8                       writeln('minore di zero')
9   else
10        if i=100 then
11                writeln('uguale a 100')
12        else
13                writeln('maggiore di 100');
```

- le righe da 1 a 13 sono un'unica istruzione if-then-else la quale ha per istruzione1 le righe da 2 a 8 e per istruzione2 le righe da 10 1 3
- le righe da 2 a 8 sono un'unica istruzione if-then-else la quale ha per istruzione1 la riga 3 e per itruzione2 le righe da 5 a 8
- le righe da 5 a 8 sono un'unica istruzione if-then-else la quale ha per istruzione1 la riga 6 e per istruzione2 la riga 8
- le righe da 10 a 13 sono un'unica istruzione if-then-else la quale ha per istruzione1 la riga 11 e per istruzione2 la riga 13

8.2 Variabili Di Tipo Boolean

Un'espressione logica genera come risultato un valore logico: vero o falso. In Pascal esiste un particolare tipo di variabile che può assumere solo i valori logici, il tipo *boolean*. La dichiarazione

```
Var
A,b: boolean
```

permette di definire le variabili a, b di tipo boolean.
L'insieme di valori che può assumere una variabile di tipo boolean è limitato a due: TRUE e FALSE.

```
a:=TRUE;
```

Possiamo assegnare a una variabile logica il valore di un'altra variabile logica:

```
b:=a;
```

Più in generale a una variabile di tipo boolean possiamo assegnare il valore restituito da un'espressione logica:

```
a:= i<100;
```

```
(*Esempio di utilizzo di una variabile boolean*)
Program prova_if;
var i:integer;
   a: boolean;

Begin
        Write('Immetti un dato intero:   ');
        Readln(i);
        a:=i<100;
        if a=TRUE then
                writeln ('Minore di 100');
end.
```

Fig. 8.1 Utilizzo di una variabile boolean

Chiedersi se a è vero è lo steso che chiedersi se a è diverso da falso, quindi potevamo scrivere:

```
if a<> FALSE then
    writeln('minore di 100');
```

Il test a=TRUE corrisponde comunque al controllo effettuato per *default*, per cui avremmo anche potuto scrivere:

```
if a then
    writeln('minore di 100'),
```

Il contenuto di una variabile booleana può essere visualizzato con l'istruzione write.

```
Write(a);
```

Oppure si può inserire all'interno delle parentesi tonde un'espressione logica

```
Write(i<100);
```

Non è invece possibile immettere direttamente il valore di una variabile booleana da tastiera, per cui la presenza di un identificatore di tipo boolean tra i parametri di una read genera un errore. Questa limitazione può essere facilmente superata utilizzando un'istruzione if come di seguito descritto:

```
write ('Digita 0 per vero; 1 per falso');
readln(i);
if i=0 then
    a:=TRUE
else
    if I=1 then
        a:=FALSE
    else
        writeln('Digitazione errata')
```

Analogamente a quanto visto per le variabili intere, il Turbo Pascal mette a disposizione del programmatore funzioni predefinite che restituiscono valori booleani. Per esempio la funzione

```
Odd(n);
```

con n integer restituisce TRUE se n è dispari, FALSE se n è pari.

8.3 Espressioni logiche

Le espressioni logiche possono contenere gli operatori relazionali che servono per confrontare fra loro dei valori.

> (maggiore di)	>= (maggiore o uguale
< (minore di)	<= (minore o uguale)
= (uguale)	<> (diverso)

Per concatenare fra lor più espressioni logiche si usano gli operatori logici not, and, or, xor.
Gli operatori logici del Turbo Pascal hanno la seguente *tavola di verità*:

A	B	Not A	A and B	A or B	A xor B
Vero	Vero	Falso	Vero	Vero	Falso
Vero	Falso	Falso	Falso	Vero	Vero
Falso	Vero	Vero	Falso	Vero	Vero
Falso	Falso	Vero	Falso	Falso	Falso

Tutti gli operatori logici hanno precedenza sugli operatori relazionali. Mentre la priorità tra gli operatori logici è mostrata in fig. 8.2.

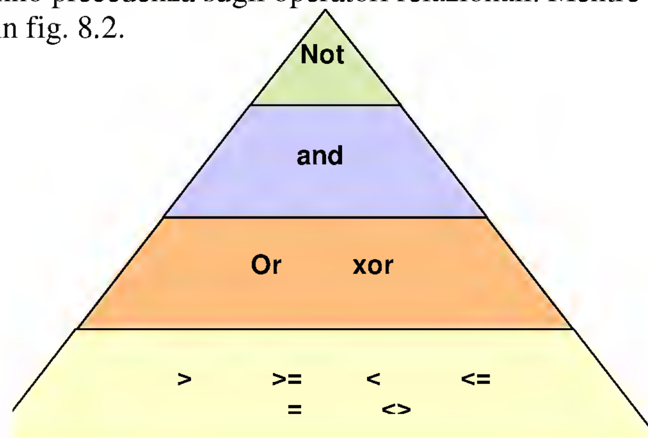


Fig. 8.2 Gerarchia degli operatori logici e booleani

8.4 Variabili di tipo carattere e codice ASCII

Le variabili di tipo carattere assumono valori alfanumerici che comprendono le lettere dell'alfabeto minuscole e maiuscole, le cifre decimali, la punteggiatura e altri simboli. Per le variabili di tipo char viene riservato in memoria lo spazio di un byte. Questo permette di fare riferimento a 256 distinte configurazioni di otto bit che individuano ogni carattere del codice in uso sulla macchina. Per assegnare un valore costante a una variabile char lo si deve racchiudere tra singoli apici:

x:='A';

A ogni carattere presente nel codice corrisponde una rappresentazione numerica univoca, per cui è possibile confrontare due simboli per mezzo di operatori relazionali. Le funzioni standard succ e prec che abbiamo applicato ai tipi integer possono essere utilizzate anche con i char.. La funzione ord permette invece di ottenere il corrispondente decimale di un carattere. Per esempio

Ord('B')=42
Succ('B')='C'
Prec('B')='A'

La funzione chr(i) ha l'effetto inverso di ord(car) e produce il carattere corrispondente all'intero i. Perciò

chr(42)='B'

Una funzione predefinita che si applica solo ai caratteri è invece upcase (maiuscola) che trasforma una lettera dell'alfabeto minuscola nella corrispondente maiuscola, mentre lascia inalterati tutti gli altri caratteri.
I valori di tipo char possono essere visualizzati con l'istruzione write e possono essere letti per mezzo dell'istruzione read.

8.5 Istruzione case

Le decisioni a più di due scelte possono essere risolte utilizzando più if-then-else in cascata. In alcuni casi una soluzione alternativa è data dal costrutto case che consente di implementare decisioni multiple basandosi sul confronto fra il risultato di un'*espressione* (integer o char) e un insieme di valori costanti:

```
case espressione of
     lista_di_costanti1:  istruzione1;
     lista_di_costanti2:  istruzione2;
     ...
[else istruzioneN;]
end;
```

Esempio:

```
Case carattere of
   'a'..'z', 'A'..'Z':   writeln('carattere');
   '0'..'9':             writeln('cifra');
   '+', '-', '*', '/':   writeln('operatore');
   else
      writeln ('carattere speciale');
   end;
```

9 STRUTTURE DI CONTROLLO ITERATIVE

9.1 Istruzione for

Se si vuole ripetere una serie di operazioni per un determinato numero di volte, si può usare la struttura iterativa for. Esempio, si vuole sommare 3 volte alla variabile somma, inizialmente posta a zero, il valore 7:

```
somma:=0;
for i:=1 to 3 do
    somma:=somma+7;
```

L'istruzione for assegna 1 alla variabile i. L'operazione i:=1 è detta *inizializzazione* e non verrà più eseguita. Ad ogni iterazione verrà eseguita l'istruzione somma:=somma+7 e verrà incrementata la variabile i di 1. Il ciclo viene iterato finchè i è minore o uguale a 3.

Più in generale il formato del costrutto for è il seguente:

```
for variabile:= valore_iniziale to valore_finale do
istruzione;
```

Come nel costrutto if, anche nel for la sintassi prevede che istruzione possa essere un blocco nel qual caso deve iniziare con la parola chiave begin e terminare con end. Il valore iniziale con cui viene inizializzata la variabile può essere un integer, positivo ma anche negativo. Per esempio:

```
for i:=1 6 to 20 do
```

esegue 5 iterazioni. Analogamente si comporta

```
For i:=-20 to –16 do.
```

Il ciclo

```
for i:=5 to 5 do
```

viene eseguito una sola volta, mentre

```
for i:=4 to 5 do
```

non viene mai eseguito.

Esiste anche una variante al costrutto for:

```
for variabile:= valore_iniziale downto valore_finale do
istruzione;
```

nel quale il valore della variabile che controlla il ciclo viene decrementata di una unità ad ogni successiva iterazione. Così con

```
For i:=5 downto 1 do
```

Il ciclo viene ripetuto 5 volte.

9.2 Istruzione while

Anche l'istruzione while, come il for, permette di eseguire ciclicamente un'istruzione:

```
while espressione_logica do
        istruzione;
```

Viene verificato che espressione_logica sia vera, nel qual caso viene eseguita istruzione. Il ciclo si ripete finchè espressione_logica risulta vera e termina quando espressione_logica diventa falsa. Istruzione_logica può essere un'istruzione composta. Mostriamo un pezzo di codice che calcola la somma di 5 numeri interi inseriti da tastiera e confrontiamo il codice realizzato con il costrutto while con il codice realizzato con il costrutto for .

Con while

```
i:=1;
while i<=5 do
begin
        write('Inser. Intero:  ');
        readln(numero);
        somma:=somma+numero;
        i:=i+1;
end;
```

Con repeat

```
for I:=1 to 5 do
begin
    write('Inser. Intero:  ');
    readln(numero);
    somma:=somma+numero;
end;
```

Come si vede, l'istruzione for gestisce automaticamente l'inizializzazione e l'incremento della variabile che controlla il ciclo mentre nel while devono essere opportunamente specificati dal programmatore.

Ogni istruzione for può essere sostituita con un'istruzione while, non è però vero il viceversa: utilizzandol'istruzione while è infatti possibile effettuare anche un ciclo in cui non è noto a priori il numero delle volte che si esegue istruzione.

Supponiamo per esempio di voler modificare il programma precedente in modo che la quantità di numeri in ingresso non sia determinata, ma termini solo quando viene inserito il valore 0.

```
(*Calcola la somma dei valori interi passati dall'utente. Termina quando viene immesso 0 (zero) *)
program somma
var
        somma, numero:integer;
begin
        writeln("Inserire 0 per terminare');
        writeln;
        numero:=1;
        somma:=0;
        while numero<>0 do
        begin
                write('Inser.intero:   ');
                readln(numero);
                somma:=somma+numero;
        end;
        writeln('Somma:   ',somma);
        readln;
end.
```

Fig. 9.1 Ciclo con numero di iterazioni non determinato a priori

In questo caso non è evidentemente possibile risolvere il problema con un for in quanto il numero dei valori immessi non è noto, ma viene deciso dall'utente solo durante l'esecuzione del programma. Inoltre nell'istruzione while, a differenza del for, la parte espressione_logica non necessariamente è del tipo i<=n, ma può essere una qualsiasi espressione logica ottenuta componendo con operatori logici più relazioni tra loro.

Se per esempio si desidera che il numero dei valori passati in ingresso non sia superiore a 10, si può inserire una variabile che conta il numero degli inserimenti e controllarne il valore all'interno del while:

while (numero<>0) and (i<=10) do....

Supponiamo che oltre alla somma si desideri determinare il valore maggiore della sequenza di ingresso, con la limitazione che i valori debbano essere tutti positivi. Il programma è quello in fig.9.2.

```
(Somma e massimo dei valori in ingresso. Termina quando viene immesso 0 o quando si sono
immessi 10 valori *)
program somma_max;
var
        somma, numero, massimo, i:integer;
begin
        writeln(''Inserire 0 per terminare');
        writeln;
        numero:=1;
        somma:=0;
        massimo:=0;
        while (numero<>0) and (i<=10)  do
        begin
                write('Inser.intero positivo:   ');
                readln(numero);
                if numero>massimo then
                        massimo:=numero;
                somma:=somma+numero;
                i:=i+1
        end;
        writeln('Somma:   ', somma);
        writeln('Massimo:   ', massimo);
        readln;
end.
```

Fig. 9.2
Calcolo della somma e del maggiore

Affinché il programma precedente operi sui numeri negativi, dobbiamo inizializzare la variabile massimo al minimo valore accettato da una variabile intera: -32768. Il Turbo Pascal mette a disposizione dell'utente una costante predefinita MAXINT, che contiene il massimo valore intero rappresentabile con una variabile di tipo integer (32767). Dato che andiamo a considerare tutti i valori interi rappresentabili, non ha più senso dare allo 0 il significato di fine sequenza in ingresso. E' perciò preferibile richiedere esplicitamente all'utente se desidera o meno terminare l'inserimento. Il listato è in fig.9.3.

(Somma e massimo dei valori in ingresso. L'utente decide esplicitamente di terminare l'immissione dei valori *)

```
program somma_max2;
var
        somma, numero, massimo, i:integer;
        ancora:char;
begin
        writeln;
        ancora:='S';
        somma:=0;
        massimo:= -MAXINT-1;
        i:=1;
        while (upcase(ancora)='S') and (i<=10)  do
        begin
                write('Inser.intero :  ');
                readln(numero);
                if numero>massimo then
                        massimo:=numero;
                somma:=somma+numero;
                i:=i+1;
                writeln(Vuoi continuare (S/N)? ');
                readln(ancora);
        end;
        writeln('Somma:  ', somma);
        writeln('Massimo:  ', massimo);
        readln;
end.
```

Fig. 9.3 Calcolo della somma e del maggiore

Utilizzando il costrutto while è importante fare molta attenzione alle iterazioni infinite che si verificano quando non si presenta mai la condizione di fine ciclo.

9.3 Istruzione repeat-until

Quando l'istruzione compresa nel ciclo deve in ogni caso essere esguita almeno una, risulta più comodo utilizzare al posto di while il costrutto:

```
repeat
        istruzione
until espressione_logica
```

Con questo costrutto viene comunque eseguita istruzione e, se successivamente espressione_logica risulta essere falsa, il ciclo viene ripetuto. Si esce dal ciclo quando la condizione dell'until diventa vera. Come sempre l'iterazione può coinvolgere un'istruzione composta. In fig. 9.4 l'esercizio precedente viene risolto con repeat-until.

(Somma e massimo dei valori in ingresso. L'utente decide esplicitamente di terminare l'immissione dei valori _*)

```
program somma_max3;
var
        somma, numero, massimo, i:integer;
        ancora:char;
begin
        writeln;
        somma:=0;
        massimo:= -MAXINT-1;
        i:=1;

        repeat
                write('Inser.intero :  ');
                readln(numero);
                if numero>massimo then
                        massimo:=numero;
                somma:=somma+numero;
                i:=i+1;
                writeln(Vuoi continuare (S/N)? ');
                readln(ancora);
        until (upcase(ancora)<>'S') or (i>10)  ;
        writeln('Somma:   ', somma);
        writeln('Massimo:   ', massimo);
        readln;
end.
```

Fig. 9.4 Calcolo della somma e del maggiore

E' importante notare come il costrutto repeat-until non necessita del begin-end al suo interno per delimitare il corpo del ciclo.

10 VARIABILI DI TIPO REAL

I numeri che hanno una parte decimale come

152.23
-91.64
0.867

non possono essere memorizzati nelle variabili di tipo integer. Le variabili che contengono tali valori sono di tipo real. La parola chiave real specifica variabili con notazione in *virgola mobile.* La dichiarazione fa sì che venga riservato uno spazio in memoria di 6 byte (48 bit) sufficiente per contenere valori positivi e negativi compresi tra 10^{-38} e 10^{38}.
Le seguenti istruzioni assegnano valori a variabili real

x:=152.23
y:=0.0008
z:=7E+20

Normalmente un real viene visualizzato da un'istruzione write o writeln con *notazione scientifica* (in formato esponenziale). Inoltre è possibile specificare il numero di cifre con cui stampare il valore. Esempio: writeln(x:15:4) riserva 15 cifre totali di cui cinque riservati alla parte decimale.

Se x=-152.23 si avrà:

Comando	*Visualizzazione prodotta*
writeln(x);	-1.522300000E+02
writeln(x:0:0);	-152
writeln(x:5:0);	-152
writeln(x:5:5);	-152.2300
writeln(x:8:1);	-152.2
writeln(x:8);	-1.5E+02
writeln(x:10);	-1.522E+02

Il tipo real viene anche utilizzato per estendere l'intervallo dei numeri interi rappresentabili.
Gli operatori aritmetici che lavorano su variabili reali sono: +, -, *, /. Le funzioni abs, sqr, random viste per gli integer, sono applicabili anche ai real. La funzione random può essere applicata in questo caso senza parametri e ritorna valori compresi tra 0 e 1. Funzioni applicabili solo ai real sono:

Funzione	*Significato*
Round(x)	Arrotonda x all'intero più vicino
Trunc(x)	Tronca x alla sua parte intera
Frac(x)	Restituisce la parte frazionaria di x (Es: frac(113.01)=0.01)
Int(x)	Restituisce la parte intera di x
Sin(x)	Seno di x, con x espresso in radianti
Cos(x)	coseno di x, con x espresso in radianti
Arctan(x)	arcotangente di x, arco in radianti la cui tangente è x
Ln(x)	Logaritmo naturale (in base *e*) di x
Exp(x)	*e* elevato alla x
Sqrt(x)	Radice quadrata di x

11 VETTORI E MATRICI

Quando si ha la necessità di trattare un insieme omogeneo di dati, esiste una soluzione diversa da quella di utilizzare tante variabili dello stesso tipo: definire un *array*, ovvero una variabile strutturata dove è possibile memorizzare più valori tutti dello stesso tipo.

Intuitivamente un *array monodimensionale* o *vettore* può essere immaginato come un contenitore suddiviso in tanti scomparti quanti sono i dati che si vogliono memorizzare. Ognuno di questi scomparti, detti *elementi* del vettore, contiene un unico dato ed è individuato da un numero progressivo, detto *indice*, che specifica la posizione dell'elemento all'interno del vettore. Il numero complessivo degli elementi viene detto *lunghezza*.

S	P	O	T
1	2	3	4

In definitiva un vettore è una struttura di dati composta da un numero determinato di elementi, tutti dello stesso tipo ognuno dei quali è individuato da un indice specifico. E' ora chiaro perché i vettori si dicono *variabili strutturate* mentre all'opposto tutte le variabili semplici siano anche dette *non strutturate*. Il tipo di dati contenuti nel vettore viene detto *tipo* dl vettore.

La sintassi per dichiarare una variabile di tipo array è la seguente:

```
a: array[1..6] of integer
```

che dichiara la variabile a di tipo array di lunghezza 6 i cui elementi sono integer.

Per accedere a un singolo elemento di a, si usa la notazione a[i] dove i è l'indice dell'elemento cui si vuole accedere. In generale un singolo elemento dell'array può essere usato come esattamente come una variabile semplice.

Si può scrivere

```
a[1]:=34
a:= b+a[1]*a[5]
```

Per loro stessa natura i vettori vengono spesso trattati all'interno di iterazioni.

```
(* Inizializzazione dell'array *)
for i:=1 to 6 do begin
   write('Inser. Intero:   ');
   readln(a[i]);
end;
```

Sono dichiarazioni valide anche:

```
variazione:array[100..149] of real;
vero[0..300] of boolean;
regola array[-23.. –11 of char;
delta... array[-5..7] of integer;
```

In tutti gli esempi trattati il numero di elementi è noto fin dall'inizio e resta costante per tutto il corso dell'eleaborazione. Tuttavia in molti problemi pratici, questo numero non è noto a priori o si modifica nel corso dell'elaborazione.In questi casi si utilizza una lunghezza maggiore al numero di dati considerati e lasciare eventualmente alcuni elementi non utilizzti. Occorre tuttavia sottolinerare che ogni elemento del vettore, anche vuoto, una volta definito, occupa uno spazio in memoria.

11.1 Esempi di uso dei vettori

Per determinare l'abilità di n concorrenti sono state predisposte due prove, entrambe con una valutazione che varia da 1 a 10. il punteggio totale di ogni concorrente è dato dalla media aritmetica delle due prove. Si richiede la visualizzazione di una tabella che contenga su ogni linea i risultati parziali e il punteggio finale del concorrente. In fig.11.1 è mostrato il listato del programma.

```
{Inizializza  i punteggi di n concorrenti su due prove
  Determina i risultati finali                              }

Program punteggi;

const
MAX_CONC =1000 ;
MIN_PUN =1;
MAX_PUN =10 ;

Var
prova1, prova2, finale :  array [1..MAX_CONC] of real ;
i, n :  integer ;

begin
repeat
        writeln ;
        write ('Numero concorrenti:    ');
        readln(n);
until (n>=1) and (n<=MAX_CONC);

{Richiesta punteggio di ogni concorrente nelle due prove}
for i:=1 to n do begin
        writeln;
        writeln('Concorrente n. ', i);

        repeat
                write ('  Prima prova:    ');
                readln(prova1[i]);
        until (prova1[i]>=MIN_PUN) and (prova1[i]<=MAX_PUN);

        repeat
                write ('  Seconda prova:    ');
                readln(prova2[i]);
        until (prova2[i]>=MIN_PUN) and (prova2[i]<=MAX_PUN);
end;

{Calcolo media per concorrente}
for i:=1 to n do
        finale[i] := (prova1[i] + prova2[i])/2;
writeln;
writeln('            Risultati Finali');
writeln;
for i:=1 to n do
        writeln(i, '° Conc. :    ', prova1[i]:7:2, prova1[i ]:7:2, finale [i]:7:2);
readln;
end.
```

Fig. 11.1 Esempio di utilizzo di un vettore

11.2 Dichiarazioni di tipo

Fino a questo punto abbiamo studiato i tipi semplici integer, char, real e il tipo strutturato array; tutti quanti predefiniti nel linguaggio. Adesso introduciamo il concetto *di tipo definito dall'utente* che ci permette di fare un salto di qualità nella scrittura dei programmi.

Un *tipo* è un insieme di valori e di operazioni che possono essere svolte su quei valori. E' possibile dichiarare un nuovo tipo, sulla base dei tipi precedentemente definiti, con la seguente sintassi:

```
type nome_tip =tipo;
```

dopo di che è possibile dichiarare una variabile di tipo nome_tipo.

Per esempio dopo la dichiarazione

```
type Tvoto=array[1..6] of integer;
```

è possibile scrivere

```
var voti: vettore;
```

Nell'esempio precedente abbiamo dichiarato:

```
prova1, prova2, finale : array [1..MAX_CONC] of real ;
```

ma potevamo scrivere:

```
type Tclassifica = array [1..MAX_CONC] of real ;
var prova1, prova2, finale : Tclassifica ;
```

Interessante è la possibilità di dichiarare dei tipi detti subrange sulla base di un sottointervallo di un altro tipo precedentemente definito. La dichiarazione di un tipo subrange è:

type nome_tipo = costante..costante;

dove costante appartiene a un tipo, come integer o char, enumerabile ovvero un tipo in cui è possibile disporre gli elementi in una determinata successione in modo che di ogni elemento sia stabilito il predecessore e il successore. Per esempio dopo aver definito

```
type eta =1..200
```

si possono dichiarare le variabili di tipo eta

```
var anni: eta;
```

per cui la variabile anni può assumere valori integer compresi tra 1 e 200; mentre con la dichiarazione

```
type lettera = 'f'..'s' ;
var c : lettera ;
```

la variabile c può assumere solo valori char compresi fra f e s.

Non è possibile definire un tipo subrange a partire da un real perché questo non è considerato un tipo enumerato dal Turbo Pascal.

11.3 Array multidimensionali

E' possibile definire array strutturati secondo più dimensioni. Per esempio in una matrice o array bidimensionale, i dati sono organizzati per righe e per colonne, come se fossero inseriti in uno schema analogo a quello della tabellina pitagorica. Per la sua memorizzazione si utilizza un array di array specificando il numero di componenti per ciascuna delle due dimensioni ch la costituiscono:

```
mat: array[1..4] of arra[1..3] of integer;
```

oppure in maniera più sintetica

```
mat: array[1..4, 1..3] of integer;
```

La variabile mat che abbiamo dichiarato contiene 4 righe e 3 colonne. Per accedere a ciascun elemento della matrice si utilizzano due indici; il primo specifica la riga, il secondo la colonna:

Per esempio mat[2, 3] fa riferimento all'elemento presente nella seconda riga della terza colonna.

Scriviamo un programma che richiede all'utente i valori da inserire, li memorizza nella matrice e li visualizza (fig.11.2).

```
{Inizializzazione e visualizzazione di una matrice}

program matrice;
var
        mat: array[1..4, 1..3] of integer;
I, j :  integer;

begin

for i:=1 to 4 do
        for j:=1 to 3 do begin
                write('Inserisci valore riga  ', i, 'colonna  ', j, ':  ');
                readln(mat[i, j]);
        end;

for i:=1 to 4 do begin
        writeln;
        for j:=1 to 3 do
                readln(mat[i, j]:5);

end;
readln;
end.
```

Fig. 11.2
Utlizzo di matrici

12 RICERCA ORDINAMENTO E FUSIONE

12.1 Introduzione

E' esperienza comune che nella gestione dei più svariati insiemi di dati sia spesso necessario:

- stabilire se un elemento è o meno presente nell'insieme;
- ordinare in modo determinato l'insieme;
- fondere due o più insiemi in un unico insieme evitando possibili duplicazioni

Queste tre attività che in informatica vengono indicate rispettivamente con i termini di ricerca, ordinamento e fusione sono estremamente frequenti e svolgono un ruolo della massima importanza in tutti i possibili impieghi dei calcolatori. E' quindi ovvio come sia della massima importanza disporre di algoritmi che svolgano questi compiti nel minor tempo possibile.

12.2 Ricerca completa

Questo algoritmo di ricerca scandisce sequenzialmente tutti gli elementi del vettore confrontandoli con il valore cercato. Nel momento in cui tale verifica dà esito positivo, la scansione termina e viene restituito l'indice dell'elemento cercato. Tale algoritmo deve controllare tutti gli elementi fino all'ultimo prima di poter concludere che l'elemento cercato non è presente nel vettore. Questo tipo di ricerca, sebbene semplice, non è il massimo dell'efficienza.

12.3 Ordinamento per selezione

Un problema collegato con la ricerca è l'ordinamento. Il più semplice algoritmo di ordinamento è l'*ordinamento per selezione*. Nella prima esecuzione viene confrontato il primo elemento del vettore con ognuno dei successivi, scambiando il valore dei due elementi se vet[1]>vet[i]. Tale procedimento si ripete per il secondo elemento del vettore il terzo etc.

```
{Ordinamento per selezione}
program ordinamento_per_selezione;
const
        MAX_ELE =1000;
type
        Tvettore = array[1..MAX_ELE] of integer;
var
        vet: Tvettore;
        I, j, n : integer;
        appoggio: integer;
begin
{Immissione lunghezza sequenza}
repeat
        write('Numero elementi:   ');
        readln(n);
until (n>=1) and (n<=MAX_ELE);
{Immissione elementi della sequenza}
for i:=1 to n do begin
        write('Immettere  ', I, ' ° elemento:  ');
        readln(vet[i]);
end;
{Ordinamento}
for j:=1 to (n-1) do
        for i:= (j+1) to n do
                if vet[j]>vet[i] then begin
                        {Scambio valori}
                        appoggio:=vet[j];
                        vet[j]:=vet[i];
                        vet[i]:=appoggio;
                end;
{Visualizzazione del vettore ordinato}
writeln;
for i:=1 to n do
```

```
        writeln(vet[i]);
readln;
end.
```

12.4 Ricerca binaria

Quando un vettore risulta ordinato, la ricerca di un valore al suo interno può avvenire mediante criteri particolari, il più noto dei quali è la così detta *ricerca binaria*.

Con questo algoritmo

- si sceglie l'elemento centrale del vettore e si confronta con l'elemento da cercare
- se si trova ok altrimenti
  - se è minore si ripete la ricerca sulla prima metà del vettore
  - se è maggiore si ripete la ricerca sulla seconda metà del vettore

```
Program ricerca_binaria;
const
        L=8;
type
        Tvettore=array[1..L] of char;
var
        vet: Tvettore;
        i,alto, basso, pos : integer;
        elemento, appoggio: char;
write('Elemento da cercare:  ');
readln (elemento);
n:=L;
alto:=1; basso:=n;
repeat
        i:= (alto+basso) div 2;
        if vet[i] = elemento then pos:=i
        else
                if vet[i]<elemento then
                        alto:=i+1
                else
                        basso:=i-1;
until (alto>basso) or (pos<>i);
writeln;
if pos<>-1 then
        write ('Elemento  ', elemento, '  presente in posizione  ', pos);
else
        write ('Elemento non presente!');
readln;
end.
```

La ricerca appena eseguita presuppone che il vettore sia già ordinato.

12.5 Fusione

Un altro algoritmo interessante è quello che partendo da due vettori ordinati ne ricava un terzo pure ordinato e contenente tutti i loro elementi. Provare a realizzare l'algoritmo che a partire da due vettori vet1 e vet2 ordinati, ricavi vet3 come fusione di vet1 e vet2:

Vet1	Vet2	Vet3
f	a	a
g	c	c
m	o	f
s		g
z		m
		o
		s
		z

13 TIPI DI DATO

13.1 Tipi e variabili

Abbiamo già definito un tipo come un insieme di valori e delle operazioni che su questi valori possono essere effettuati; una variabile è invece un'entità usata per memorizzare un valore a cui appartiene. I tipi di dati in Pascal sono raffigurati in fig.13.1.

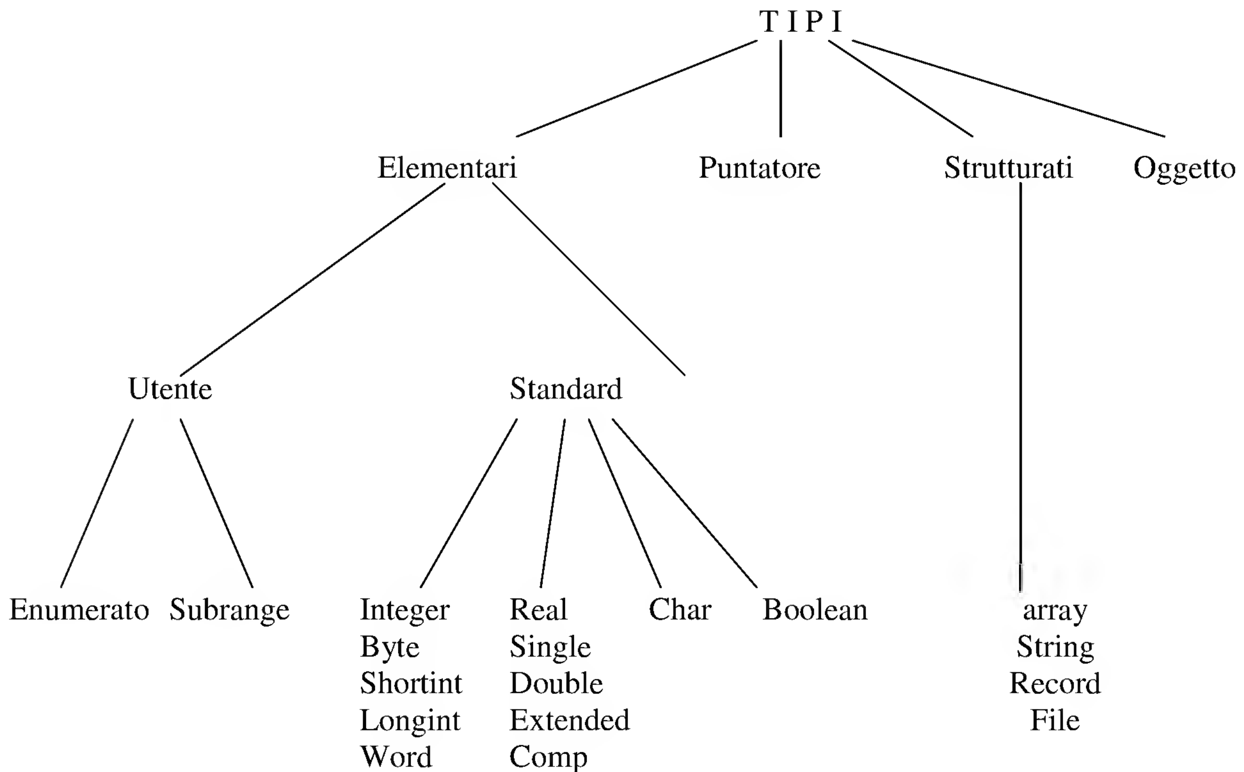


Tabella 9.1
Tipi numerici interi

Tipo	Intervallo	Memoria occupata (in byte)
Integer	-32768 ÷32767	2
Byte	0÷255	1
Shortint	-128÷127	1
Longint	-2147483648÷ 2147483647	4
word	0÷65535	2

Tipo	Intervallo	Memoria occupata (in byte)
Real	$2.9 \cdot 10^{-39} \div 2.9 \cdot 10^{38}$	6
Single	$1.5 \cdot 10^{-45} \div 3.4 \cdot 10^{38}$	4
Double	$5.0 \cdot 10^{-324} \div 1.7 \cdot 10^{308}$	8
extended	$3.4 \cdot 10^{-4932} \div 1.1 \cdot 10^{4932}$	10
Comp	$0 \div 2.9 \cdot 10^{18}$	8

Tabella 9.2
Tipi numerici reali

13.2 Tipi definiti dall'utente

Tra i tipi definiti dall'utente troviamo il tipo *enumerato* per il quale devono essere esplicitamente specificati i valori che può assumere: La definizione è la seguente:

```
type nome_tipo = (valore1, valore2, valore3, …, valoreN);
```

Esempi:

```
type
sesso=(m, f);
nota=(do, re, mi, fa, sol, la, si);
colore=(rosso, giallo, verde);
mese=(gennaio, febbraio, marzo);
var
        persona:sesso;
        accordo:nota;
        semaforo:colore;
        scadenza:mese;
```

L'ordinamento all'interno di una enumerazione è definito, a partire da zero, dalla sequenza in cui compaiono gli elementi. Come per tutti gli altri ordinali è quindi possibile applicare all'interno di un tipo enumerato le funzioni ord, succ e pred.

```
Ord(giallo)=1;
Succ(giallo)=verde;
prec(giallo)=rosso;
```

Inoltre i numerali hanno un ordinamento di tipo ciclico:

```
succ(verde)=rosso;
```

Non è invece possibile stampare direttamente il valore di una variabile di tipo enumerato scrivendo per esempio write(semaforo). Per ottenerlo si devono usare degli artifizi, del tipo:

```
case semaforo of
        rosso: write('rosso');
        giallo: write('giallo');
        verde: write('verde');
end;
```

Analogamente non si può assegnare un valore alla variabile tramite lettura con istruzioni del tipo read(semaforo). Perciò si è costretti a usare variabili ausiliarie e superare il problema con costrutti come

```
Write ('Introduci il mese dell''anno in cifre:  ');
case n of
        1: scadenza:=gennaio;
        2: scadenza:=febbraio;
        3:scadenza:=marzo;
end;
```

Il tipo subrange è definito sulla base di un intervallo di variabilità di un tipo precedentemente definito, detto tipo scalare associato. Il tipo scalare associato non può essere di tipo reale (real, single, double, extended, comp).
Esempi:

```
type
        giorno =1..7;                  {subrange degli integer}
        alfabeto = 'a'..'z';           {subrange dei char}
        estivi = giugno..settembre;{subrange di mese}
```

14 RECORD E TABELLE

Il tipo strutturato record permette l'aggregazione di informazioni di tipo diverso. Al contrario di quanto accade per gli array si tratta di un aggregato di dati non omogeneo. Ognuna delle informazioni del record è detta campo.

```
Type
Nome_record = record
        Nome_campo: tipo_campo;
        Nome_campo: tipo_campo;
        ....
        Nome_campo: tipo_campo;
end;
```

Ogni campo è costituito da un nome e da un tipo, il quale può essere uno qualsiasi tra quelli ammessi, anche un altro record precedentemente definito o un array. Il numero dei campi non è limitato.

Esempio:

```
type
        Tpersona = record
                nome: string[40];
                anni: integer;
                residenza: string[50];
        end;
var persona:Tpersona;
```

Per fare riferimento a un campo del record si deve far seguire il nome della variabile da un punto e dal nome del campo.

Esempi di istruzioni:

```
persona.nome:='Paolo';
persona.anni:=20;
read(persona.nome);
write (persona.nome);
```

14.1 Clausola with

Quando all'interno di un programma si deve citare più volte il nome di una variabile di tipo record si può utilizzare l'istruzione with per riferirsi in modo più diretto ai suoi campi:

```
with nome_record do istruzione
```

All'interno di istruzione, che può essere un blocco begin-end, i nomi dei campi potranno non essere preceduti dal nome del record.

Esempio:

```
        with persona do
                nome:='Paolo';
                anni:=20;
        end;
```

14.2 Tabelle e chiavi

Gli array composti da elementi di tipo record sono conosciuti con il nome di tabelle. Per esempio un'agenda può essere definita mediante le seguenti dichiarazioni:

```
type
        persona = record
                nome  : string[40];
                cognome: string[40];
                telefono: string[40];
        end;
var
        agenda : array [1..5] of integer;
```

	Nome	Cognome	Telefono
a[1]	Anna	Calza	76543
a[2]	Mario	Chiari	99876
a[3]	Tommaso	Conforti	11244
a[4]	Anna	Cipolla	98123
a[5]	Matilse	Costantino	44533

Analogamente all'inizializzazione di un vettore, anche l'inizializzazione di una tabella si effettua attraverso un ciclo, al cui interno si inizializzano campo per campo i vari record della tabella. In riferimento a una tabella si utilizza il termine *chiave* per indicare un qualsiasi insieme di campi che permetta di individuare in modo univoco i record.

14.3 Gestione di una tabella

Realizziamo un programma che gestisce un'agenda telefonica, in particolare consente di effettuare inserimenti nell'agenda e visualizzazione dei dati in essa contenuti.

```
Program gestione_agenda
{Gestione di un'agenda di nomi e numeri di telefono per mezzo di un menu di opzioni scelte
dall'utente.}
uses crt;
type persona =record
        nome: string[40];
        cognome: string[40];
        telefono: string[10];
end;

var
        agenda: array[1..100] of persona;
        lung_agenda: integer;
        scelta, i: integer;
        fine_ins:char;
begin
        clrscr;
        lun_agenda:=0;
        scelta:=-1;
        while scelta <>0 do begin
                clrscr;
                gotoxy(25,8); write ('GESTIONE AGENDA');
                gotoxy(26,10); write ('1. Aggiungi');
                gotoxy(26,11); write ('2. Visualizza');
                gotoxy(25,12); write ('0. Fine');
                gotoxy(28,19); write ('Scegliere un''opzione:  ');
                readln(scelta);
                clrscr;
                i:=lung_agenda+1;
                case scelta of
                        1: repeat {Aggiunge nuovi nominativi}
                                with agenda[i] do begin
                                        write('nome:  ');
                                        readln(nome);
                                        write('cognome:  ');
                                        readln(cognome);
                                        write('telefono:  ');
                                        readln(telefono);
                                end;
                                lung_agenda:=lung_agenda+1;
                                writeln ('Vuoi inserire un altro nominativo? (S/N)  ');
                                readln(fine_ins);
                          until upcase(fine_ins)='S' ;

                        2:  begin   {Visualizza i nominativi in agenda}
                                clrscr;
                                for i:=1 to lung_agenda do
                                        writeln(agenda[i].nome, '     ', agenga[i].cognome, '     ',
                                                agenda[i].telefono);
                                writeln;
                           end;
                end;
        end;
end.
```

14.4 Limiti delle tabelle

Come tutte le strutture di tipo array, anche le tabelle contengono un numero prefissato e immutabile di elementi. Questo implica che nel corso dell'elaborazione non è possibile inserire un numero arbitrario di elementi ma saremo vincolati a non superare quella che è la dimensione massima dell'array. D'altronde, non è neppure opportuno sovrastimare eccessivamente la dimensione della tabella in quanto gli elementi che la compongono occupano comunque uno spazio in memoria, anche se non contengono alcun dato significativo.

Osserviamo inoltre come l'inserzione in una tabella ordinata di un nuovo record, non mantenga alcun tipo di ordinamento. Verranno esaminati delle strutture dati più efficienti per risolvere entrambi questi problemi.

Inoltre le tabelle, come tutte le altre variabili, sia semplici che strutturate, non sono permanenti in memoria ma vengono automaticamente cancellate al termine del programma; così che quando questo viene mandato in esecuzione non si hanno più a disposizione i dati inseriti dall'utente nella precedente elaborazione. Questo problema può essere risolto da strutture dati che utilizzano memorie di massa e permettono una memorizzazione permanente dei dati stessi.

15 FILE

Con il termine *file* (archivio, fila) viene indicato ogni insieme di informazioni memorizzato sulla memoria di massa.
In Pascal un *file* è costituito da una sequenza di elementi omogenei e la sua dichiarazione di tipo ha la seguente sintassi:

```
type
        nome_tipo_file = file of tipo;
```

una variabile di questo tipo sarà dunque dichiarata come

```
var
        var_file : nome_tipo_file;
```

Esempio:

```
type
        libro = record
                titolo: string[100];
                autore:string[80];
                altri_autori: array[1..4] of string;
                soggetti: array [1..8] of string;
                editore: string[80];
end;
var
        biblioteca: file of libro;
```

15.1 Apertura e chiusura di file

Abbiamo visto che per la gestione di file facciamo riferimento a un identificatore di variabile di tipo file, identificatore che viene detto *nome logico* in quanto è usato solo all'interno del nostro programma. Per iniziare a lavorare si deve associargli un *nome fisico*, quello effettivamente conosciuto dal sistema operativo. E' infatti il sistema operativo che si occupa della memorizzazione e della gestione dei file e i programmi devono quindi chiedergli, tramite le procedure e le funzioni che vedremo, i servizi di cui hanno bisogno.La prima operazione da eseguire su di un file è effettuata mediante la procedura

```
Assign(var_file, 'nome_file')
```

Che associa la variabile var_file di tipo file a un file fisico presente su memoria di massa e chiamato nome_file. Se non si lavora su Windows, nel sistema operativo MS-DOS il nome di un file fisico deve essere del tipo nome.estensione dove nome è composto al più da otto caratteri mentre estensione è opzionale e può essere composta da al più tre caratteri. Se poi il file fisico non è memorizzato nella directory corrente, c:\TP\BIN, il suo nome deve essere preceduto dal nome del *drive* e dal *path* della directory in cui si trova.

Successivamente il file, per poter essere usato, deve venire aperto. A questo scopo, se il file già esiste, si usa la procdura

Reset (var_file);

L'apertura prepara il file per le successive operazioni di lettura e scrittura. Viene infatti allocato in memoria centrale un apposito spazio, detto buffer, che avrà la funzione di area di passaggio per la lettura/scrittura su quel file, ovvero di un'area predisposta alla comunicazione fra la memoria centrale e quella di massa dove sarà memorizzato il file. Viene inoltre associato un puntatore al file con assegnato il riferimento al primo elemento. Infatti, in mancanza di ulteriori operazioni, lettura e scrittura inizieranno proprio dal primo elemento.

```
Program prova_file_testo;
type
  persona=record
          nome, cognome: string[40];
          indirizzo: string[40];
          cap: string[5];
          citta: string[40];
  end;
  anagrafe=file of persona;
var residenti: anagrafe;

begin
  assign(residenti, 'c:\documenti\elenco.dat');
  reset(residenti);
  .....
```

Nell'esempio viene associato alla variabile residenti (nome logico) di tipo anagrafe, il nome fisico elenco.dat.
Se il file non esiste o se prima di utilizzarlo si vuole cancellare quello già presente sulla memoria di massa, per aprirlo non si deve usare reset ma

Rewrite(var_file);

Attenzione: rewrite è un'istruzione distruttiva e se il file esiste già lo cancella completamente distruggendo così tutte le informazioni che contiene. Dopo aver terminato le operazioni di lettura e/o scrittura è necessario eseguire l'operazione di chiusura del file

Close(residenti)

La chiusura garantisce che tutti i dati scritti dal programma sul file residenti e momentaneamente memorizzati nel buffer in memoria centrale siano salvati su disco. Infatti, poiché le operazioni di input/output fra memoria di centrale e memoria di massa sono molto più lente rispetto all'esecuzione delle altre istruzioni, molto spesso il sistema operativo ritarda la scrittura sulla memoria di massa, immagazzinando temporaneamente le informazioni nel buffer così da ottimizzare le prestazioni del programma.
Dopo aver eseguito assign per associare alla variabile interna var_file il nome fisico di un file presente su disco, tutte le operazioni effettuate successivamente sul file come reset, rewrite, close e le altre che vedremo in seguito, faranno riferimento a quel file mediante var_file mentre il suo nome fisico non verrà più utilizzato.

15.2 Lettura e scrittura su file

Dopo l'apertura è possibile leggere i record del file, modificarli, inserirne di nuovi. A tele scopo abbiamo a disposizione le procedure read e write, con la seguente sintassi:

```
read (var_file, lista_di_variabili);
write (var_file, lista_di_variabili);
```

dove lista_di_variabili è costituita da una o più variabili, separate da virgola, dello stesso tipo del file. Per specificare la posizione all'interno del file di record da leggere o scrivere il sistema gestisce un apposito puntatore al coosidetto record corrente che la momento dell'apertura con un'istruzione reset è posizionato sul primo record del file. Dopo ogni comando read o write il puntatore al file viene automaticamente spostato sul record successivo a quelli letti o scritti. Questo record diventa così il nuovo record corrente. Se per esempio x è una variabile di tipo persona

read(residenti, x);

legge il record corrente del file residenti e inserice il suo contenuto nella variabile x mentre

write(residenti, x);

scrive il contenuto di x nel record corrente del file residenti.

Read e write sono le procedure ben note. Finora sono state usate solo nel caso in cui l'entrata (read) veniva catturata dalla tastiera mentre l'uscita (write) veniva inviata al video senza indicare var_file. Volendo però anche in questi casi usare la forma generale si può scrivere:

read(input, lista_di_variabili);
write(output, lista_di_variabili);

dove input rappresenta l'input standard e autput l'output standard: video.
Nell'intestazione del programma possiamo indicare opzionalmente I file su cui il file opererà

Program prova_file_testo(input, output, residenti)

Sono inoltre disponibili comandi per la gestione dei file. In particolare possono essere funzioni e procedure predefinite.
Le *funzioni* sono sottoprogrammi che a partire da uno o più valori presi in ingresso restituiscono un valore al programma chiamante. Le *procedure* al contrario delle funzioni, sono dei sottoprogrammi che non restituiscono alcun valore.
Le *funzioni* messe a disposizione dal Turbo Pascal per la gestione di file sono:

eof(var_file)

funzione booleana che restituisce true se il puntatore di var_file è posizionato alla fine del file, false altrimenti.

Filepos(var_file)

restituisce il numero progressivo del record corrente

Filesize(var_file)

restituisce il numero complessivo dei record contenuti nel file. Questa funzione consente di posizionarsi direttamente sull'ultimo record del file, così che è possibile sostituire il ciclo

While not (eof(a)) do read(a,p)

con l'istruzione di accesso diretto

Seek(a, filesize(a));

Le procedure messe a disposizione dal Turbo Pascal per la gestione di file sono:

Seek (var_file_n)

dove n è un'espressione intera.
La procedura seek posiziona il puntatore direttamente sull'n-esimo elemento del file. Grazie a questa procedura si realizza un accesso *diretto* al file o *casuale* (*random*) al file che consiste nel posizionarsi direttamente sull'elemento desiderato conoscendone la posizione n all'interno del file.
Poiché le operazioni di accesso al file (read e write) sono molto più costose, in termini di tempo, delle altre operazioni effettuate in memoria centrale, l'impiego di seek, filesize, filepos, riducendo il numero di accessi in memoria, garantisce un notevole risparmio di tempo.
Infile le procedure

Erase(var_file)

che cancella il file fisico dal disco e

Rename(var_file, 'nome_file')

che rinomina il file.

15.3 Cancellazione logica e fisica di record

Nei programmi per la gestione del file non abbiamo fornito una procedura per cancellare uno o più record da un file ma bisogna limitarsi a effettuare una cancellazione logica assegnando ai campi del record che si vuole cancellare, o più semplicemente alla sua chiave, una stringa bianca. In questo modo lo spazio della memoria di massa occupato dal record non viene però reso disponibile. Per questo, quando il numero dei record cancellati logicamente è diventato abbastanza grande, si provvede a ricopiare i record non cancellati, in un secondo file.
Per la cancellazione logica, si può pensare di aggiungere un campo booleano alla definizione del record che marca il record come cancellato. Vedi listato paragrafo successivo.

15.4 Gestione di un'agenda

Vediamo un esempio completo di utilizzazione di un file per la gestione di una rubrica telefonica. Il programma in fig. 15.1 inizializza il file.

```
Program inizializza_agenda ;
uses crt;
type
        Tpersona=record
                nome:string[40];
                telefono:string[40];
        end;
var
        agenda:file of persona;
        persona:persona;
        cancellato: boolean;  {serve per marcare un record come cancellato}

begin
        clrscr;
        assign (agenda, 'agenda'); {assegnamento nome logico file}
        rewrite(agenda);
end;
```

Fig 15.1 Inizializzazione di un file

Il programma in fig. 15.2 gestisce l'agenda telefonica, si suppone che il file sia già stato creata con il programma precedente.

Fig 15.2 Gestione agenda

```
Program gestione_agenda ;
uses crt;
type
        Tpersona=record
                nome:string[40];
                telefono:string[40];
                cancellato: boolean;
        end;
var
        agenda:file of persona;
        persona:persona;
        i, n, scelta: integer;
        fine_ins:char;
begin
        clrscr;
        assign (agenda, 'agenda.dat');
        scelta:=-1;
        while scelta <>0 do begin
                clrscr;
                gotoxy(25,8); write ('GESTIONE AGENDA');
                gotoxy(26,10); write ('1. Aggiungi');
                gotoxy(26,12); write ('2. Aggiorna');
                gotoxy(26,14); write ('3. Visualizza');
                gotoxy(26,16); write ('4. Cancella');
                gotoxy(26,18); write ('0. Fine');
                gotoxy(28,22); write ('Scegliere un''opzione:  ');
                readln(scelta);
                clrscr;
                case scelta of
                                1: begin
                                        reset(agenda);
                                        repeat {Aggiunge nuovi nominativi}
                                        write('nome:  '); readln(persona.nome);
                                        write('telefono:  '); readln(persona.telefono);
                                        write(agenda, persona);
                                        writeln ('Vuoi inserire un altro nominativo? (S/N)  ');
```

```
            readln(fine_ins);
         until upcase(fine_ins)='S' ;
       close(agenda);
    2:  begin
            reset(agenda);
            i:=1;
            write('Nome da ricercare:  ');
            readln(persona1.nome);
            while not (eof(agenda) and
                        (persona.nome<> persona1.nome) do begin
                read(agenda, persona);
                i:=i+1;
            end;
            if persona.nome = persona1.nome then begin
                write('     nome:  ', persona.nome,
                      '     telefono:   ', persona.telefono);
                write('Deve essere aggiornato con…');
                write('    nome:   '); readln(persona1.nome);
                write('    telefono:   '); readln(persona1.telefono);
                {posizionamento sul record da modificare}
                seek(agenda, i-1);
                {aggiornamento del record}
                write(agenda, persona1);
            end
            else begin
                write('Non presente in agenda…');
                readln;
            end;
        close(agenda);
        end;
    3:  begin   {Visualizza i nominativi }
            reset(agenda);
            clrscr;
            reset(agenda);
            while not (eof(agenda) do begin
            read(agenda, persona);
            write(persona.nome, '     ', persona.telefono);
            writeln;
            end;
        close(agenda);
         end;
    4: begin
            reset(agenda);
            i:=1;
            write('Nome da cancellare:  ');
            readln(persona1.nome);
            while not (eof(agenda) and
                        (persona.nome<> persona1.nome) do begin
                read(agenda, persona);
                i:=i+1;
            end;
            if persona.nome = persona1.nome then begin
                {posizionamento sul record da cancellare}
                seek(agenda, i-1);
                read(agenda, persona);
                persona1.nome:=persona.nome;
                persona1.telefono:=persona.telefono;
                persona1.cancellato:=true;
                {Cancellazione logica del record}
                write(agenda, persona1);
            end
            else begin
```

```
                    write('Non presente in agenda...');
                    readln;
                end;
                clrscr;
            close (agenda);
            end;
        end;
    end;
end.
```

15.5 File di tipo testo

Esiste un tipo di file predefinito, equivalente alla dichiarazione

```
Type
Text =file of char
```

Detto file di testo che presenta alcune specificità. Il programma ha a disposizione il tipo text e non deve quindi esplicitare la precedente dichiarazione.
Poiché questo tipo di file puo essere trattato solo in modo sequenziale, non è possibile utilizzare la procedura seek e le funzioni filesize e filepos. Si posono invece usare, oltre alle abituali read e write anche le procedure

```
Readln( var_file, carattere)
```

Che dopo la lettura salta all'inizio della riga successiva e

```
Writeln(var_file, carattere)
```

Che dopo la scrittura del carattere sul file vi aggiunge un indicatore di riga nuova.
Oltre a eof, che abbiamo esaminato, per i file di testo abbiamo anche la funzione booleana

```
Eoln(var_file)
```

Che restituisce true ogni volta che leggendo un file viene incontrato carattere newline.
E' poi definita la procedura

```
Append(var_file)
```

Che consente di aggiungere altro testo alla fine del file var_file già presente in memoria.
I file di tipo testo sono di estrema importanza e di uso molto frequente: per esempio i listati dei programmi scritti utilizzando l'editor del Tubo Pascal vengono salvati sul disco come file text.

```
Program testo;
uses crt;
var lettera: text;
c:char;

begin
{cattura il testo inserito dall'utente e lo inserisce in un file}
    assign(lettera, 'document.doc') ;
    clrscr;
    writeln('DIGITA IL TESTO (# per finire)');
    writeln;
    rewrite(lettera);
    read(c);
    while not (c='#') do begin
        write(lettera, c);
        read (c);
    end;
    close(lettera);
end.
```